Optimization of Search Results with Duplicate Page Elimination using Usage Data

A. K. Sharma[1], Neelam Duhan[2]
[1, 2] Department of Computer Engineering,
YMCA University of Science & Technology, Faridabad, India
[1]Email: ashokkale2@rediffmail.com
[2]Email: neelam_duhan@rediffmail.com

***Abstract*—The performance and scalability of search engines are greatly affected by the presence of enormous amount of duplicate data on the World Wide Web. The flooded search results containing a large number of identical or near identical web pages affect the search efficiency and seek time of the users to find the desired information within the search results. When navigating through the results, the only information left behind by the users is the trace through the pages they accessed. This data is recorded in the query log files and usually referred to as Web Usage Data. In this paper, a novel technique for optimizing search efficiency by removing duplicate data from search results is being proposed, which utilizes the usage data stored in the query logs. The duplicate data detection is performed by the proposed Duplicate Data Detection (D3) algorithm, which works offline on the basis of favored user queries found by pre-mining the logs with query clustering. The proposed result optimization technique is supposed to enhance the search engine efficiency and effectiveness to a large scale.**

***Index Terms*—WWW, Search Engine, Query log, Clustering, PageRank**

I. Introduction

The Web had been unparalleled in many ways- in scale, in almost complete lack of coordination in its creation, and in the diversity of backgrounds and motives of its participants. One of the essential features, which led to the explosive growth of the web, is the uploading of numerous copies of web documents on the WWW. The tremendous volume of identical or near identical copies of web documents has posed challenges to the performance and scalability of web search engines. Duplicate data detection is an inherent problem that search engines must have to deal with besides other routine functions.

It has been reported that about 10% hosts are mirrored to various extents in a study including 238,000 hosts [1]. The efficient identification of duplicates is a vital issue that has arose from the escalating amount of data and thus needs to be addressed. Duplicate documents refer not only to completely identical documents but also to nearly identical documents. Though near duplicate documents display striking similarities, they are not bit wise similar [2]. If search engines do not solve the problem of duplicate elimination effectively, generally, in response to a web search, many duplicates may appear in the results. Such duplicates will significantly decrease the perceived relevance of search engines.

In this paper, an approach has been proposed that filters the search results by eliminating duplicate pages detected by the proposed Duplicate Data Detection (D3) algorithm.

The method detects duplicates and near duplicates in an offline mode, while their elimination can be performed online by the search engines. The query logs are pre-mined by applying the query clustering techniques and the discovered query clusters are in turn utilized for finding duplicate pages. The paper has been organized as follows: Section II describes the current research carried out in this area; Section III illustrates the proposed work based on pre-mining the query logs; Section IV shows the performance of proposed work with example scenario and the last section concludes the paper.

II. Related Work

The notion of Search Result Optimization and Duplicate Data Detection has been a subject of interest since many years. For optimizing the search results, majority of the present day search engines are using the techniques of one kind or another such as document clustering, page ranking and utilization of user feedback etc. The document clustering methods has been used by many search engines [3] to automatically group the retrieved documents into a list of meaningful categories. Northern Light and Vivisimo are such type of search engines. Ranking methods [4, 5] are also being applied by search engines in order to provide a sorted list of pages to the user based on page relevance. The query logs are considered by many researchers [6] in different aspects of search engines such as building query recommendation systems, inferring semantic concepts or relations from the users' queries etc.

The proposed duplicate data detection mechanism is also made to utilize query logs, which forms the basis of proposed search result optimization system (Section III). A number of researchers have discussed the problem of finding duplicate data on the web by employing different techniques other than query logs. Some of these methods are described below.

An algorithm called *Dust Buster* [7], for uncovering DUST (*Different URLs with Similar Text*) was presented in 2007. The method intended to discover rules that transform a given URL to other URLs that are likely to have similar content. Dust Buster employs historical crawl logs or web server logs, instead of probing the page contents, to mine the dust efficiently. The benefits provided by the method are: increased crawler effectiveness, reduced indexing overhead, and improved quality of popularity statistics such as Page Rank.

A technique for estimating the degree of similarity among pairs of documents was presented in 1997, known as *shingling* [8], which does not rely on any linguistic knowledge other than the ability to tokenize the documents into a list of words. Therefore, it is merely syntactic. In shingling, all word sequences of adjacent words are extracted. If two documents contain the same set of shingles they are considered equivalent and if their sets of shingles appreciably overlap, they are declared similar to different extents.

Ilyinsky et al. [9] suggested a method of *descriptive words* for the definition of near-duplicate documents, which was based on the choice of *N* words from the index to determine a "signature" of a document. Any search engine based on the inverted index can apply this method. The methods based on "shingles" and "signature" were compared by the authors. At almost equal accuracy of algorithms, the method in the presence of inverted index was more efficient.

An approach called *Copy Detection* [10] determines the similar web documents, similar sentences and graphically captures the similar sentences in any two web documents. Besides handling wide range of documents, this copy detection approach is applicable to web documents in different subject areas as it does not require static word lists.

There are many efficient methods available for search result optimization and duplicate detection [11-15], but a critical look at the available literature indicates that none has used the concept of query logs in duplicate page removal from the search results to optimize and prune the search space and thus, enhance the search efficiency. Moreover, the duplicate detection techniques are not much scalable due to the abundance of web pages on WWW. Therefore, a mechanism needs to be introduced for efficient detection of duplicates and incorporation of the duplicate detection process in the search result optimization. A framework for duplicate data detection and elimination on the basis of mining query logs is proposed in the next section, wherein the offline method uses query log to identify duplicate data and online method filters the search results by eliminating the duplicate pages. The offline method works efficiently without affecting the throughput and performance of search engine.

III. Proposed Result Optimization Framework

A framework for search result optimization has been designed, which detects duplicate pages and eliminates them from the search results (if they happen to appear) as shown in Fig. 1. It has two main modules: the *Duplicate Data Detector* and the *Duplicate Eliminator*. The *Duplicate Data Detector* performs query based duplicate data detection offline by extracting favored or popular user queries by means of query clustering. The information about duplicates is stored in a repository called '*Duplicate_Doc Database*'. The *Duplicate Eliminator* works online by removing duplicate pages from the search results by consulting this database. The detailed working of these modules is explained in the next subsections.

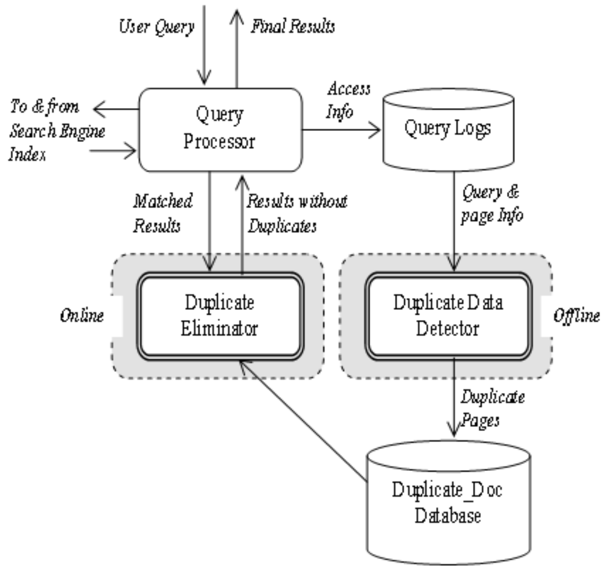


Figure 1. Duplicate Data Detection & Elimination Framework

A. Duplicate Data Detector

This module works on the basis of information extracted from the query logs as shown in Fig. 2. As outlined, initially, a sub-module referred to as *Query Cluster Generator* produces clusters of the historical user queries stored in query logs and saves them in an intermediate repository. Then, with the help of *Favored query finder*, popular user queries are identified from the query clusters. These favored queries are then executed offline and at last duplicate and near duplicate documents are extracted from the retrieved results with the help of *Duplicate Page Extractor* and stored to *Duplicate_Doc* database.

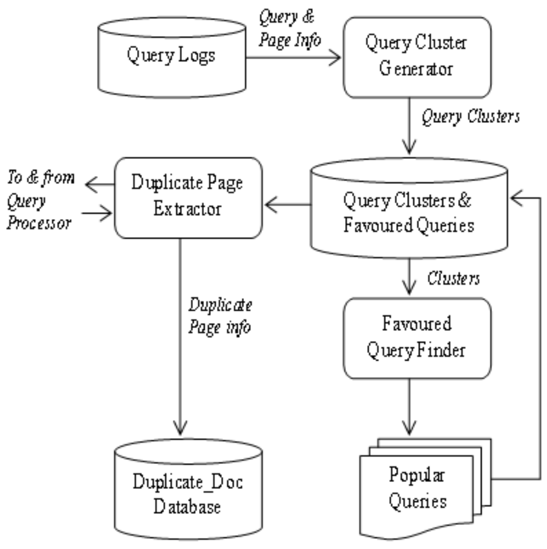


Figure 2. The Working of Duplicate Data Detector

The proposed Duplicate Data Detector works offline and consists of the following functional components:

Query Cluster Generator

This sub-module is used to generate clusters of user queries, which are stored in query logs built by search engines and for this purpose, it uses following two principles:

Principle 1 (Based on query contents): If two queries contain same or similar terms, they are supposed to denote the same or similar information needs. Keyword based similarity function is defined as follows:

$$Sim_{keyword}(qi, qi+1) = \frac{KW(qi) \cap KW(qi+1)}{KW(qi) \cup KW(qi+1)} \qquad (1)$$

where q and q_{i+1} are two user queries; $KW(q)$ denotes the number of keywords in a query q.

Principle 2 (Based on document clicks): If two queries lead to the selection of the same documents (document clicks), they are considered similar. Clicked URL based similarity function is defined as follows:

$$Sim_{clickedURL}(qi, qi+1) = \frac{URL(qi) \cap URL(qi+1)}{URL(qi) \cup URL(qi+1)} \qquad (2)$$

where $URL(q)$ denotes the number of clicked documents with respect to query q.

Both principles have been considered important to determine the similarity of queries and thus, any one of them can not be ignored; therefore, a combined measure has been defined to take advantage of both principles as is given below:

$$Sim_{combined} = \lambda \times Sim_{keyword} + \mu \times Sim_{clickedURL} \qquad (3)$$

where λ and μ are constants with $0 \leq \lambda, \mu \leq 1$ and $\lambda + \mu = 1$.

A threshold measure (T) for similarity values has been selected to find out the queries to be placed in a particular cluster. The output of Cluster Generator has been shown in Fig. 3, where each i^{th} cluster consists of similar queries, the associated keywords of queries in the set KW_i and their associated clicked URLs in the set URL_i.

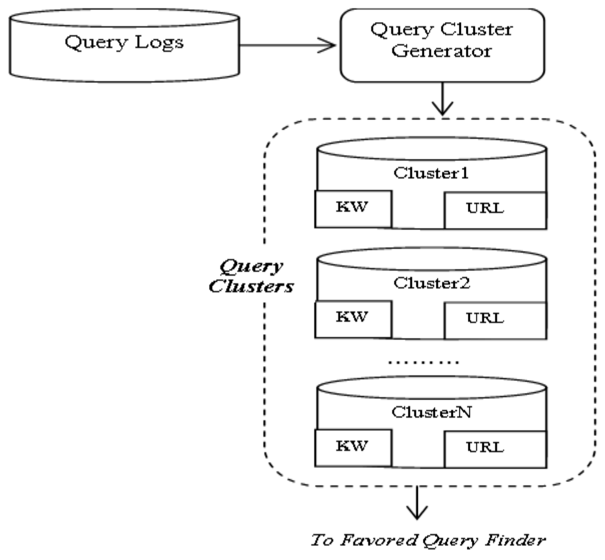


Figure 3. The output of Query Cluster Generator

The purpose of generating query clusters is to group similar queries in one cluster. When a new user query is fired, the query terms are matched with keyword set KW of different clusters and query is assigned to the matched cluster. The corresponding search results of the user query are checked for duplicate elimination with respect to only the matched cluster. The detailed algorithm of cluster generator has been explained in Fig. 4

```
Algorithm: Query_Cluster_Generator()
I/P: Similarity constants λ and μ, Similarity Score
threshold T, Query log with the following fields:
   1. IP address of the users.
   2. QueryID: each query is assigned a unique ID
   3. Query: query issued by the user and its tokens.
   4. Item rank: if the user clicked on a search result, then
      rank of the item on which he clicked.
   5. Clicked URL: if the user clicked on a search result, the
      domain position of the URL.
O/P: Set of Clusters C= {C1, C2, ...Cn}, each cluster Ci
with following information:
   1. A cluster-ID
   2. A collection of related queries.
   3. A set of cluster keywords (clustKW) and cluster URLs
      (clustURL) of queries present in the cluster.
//Start of Algorithm
For (each query $q_i$ ∈ Query_Log)
      { Flag [$q_i$]= 0;            // initially each query is unvisited
       clustURL(qi)= clustKW(qi)= ∅
      }
ClusterId= new (ClusterId);      //initialize a ClusterId say C1
For (each query $q_i$)
 {
    If ( Flag ($q_i$) = =1 ) then
       continue;     //sure that qi is not a part of any existing cluster
    else             //create a new cluster with respect to query $q_i$
      Assign qi the ClusterId;
        // Place clicked URLs & tokens of query in respective sets
      $clustURL(qi) = clustURL(qi) \cup urls(qi)$
      $clustKW(qi) = clustKW(qi) \cup tokens(qi)$
       For (each query $q_{i+1}$)
       {                  // find the similarity of two queries using (3)
          Find $Sim_{combined}(q_i, q_{i+1})$;
          If ( $Sim_{combined}(q_i, q_{i+1})$ > = T) then
             {  Assign $q_{i+1}$ the same ClusterId as of qi;
                $clustURL(qi) = clustURL(qi) \cup urls(q_{i+1})$ ;
                $clustKW(qi) = clustKW(qi) \cup tokens(q_{i+1})$ ;
             }
          Flag [qi]= 1;
          ClusterId= next (ClusterId)
                              //Initialize the next Id (C2, C3 etc.)
       } //end inner for
 } //end outer for
Return the clusters C1, C2,... Cn
```

Figure 4. Algorithm for Cluster Generator

Favored Query Finder

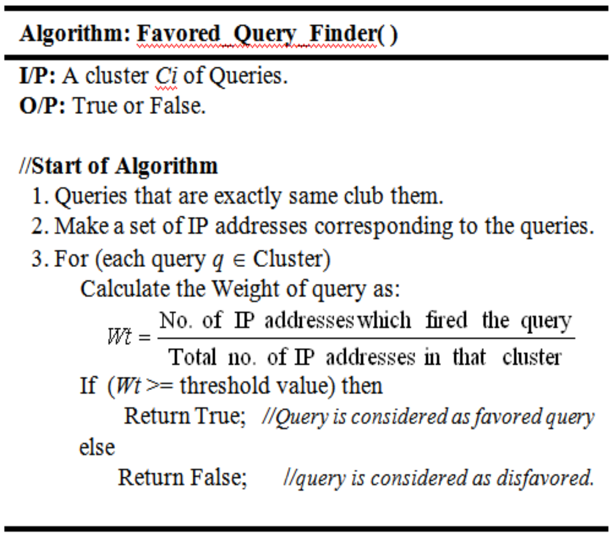
Once query clusters are formed, the next step is to find a set of favored/popular queries from each cluster. A favored query is one, which occupies a major portion of the search requests in a cluster and it is assumed that, in general, its corresponding search results tend to have more duplicates in search results than disfavored queries. The process of finding favored queries has been shown in Fig. 5.

Duplicate Page Extractor

After extracting the popular user queries from their respective query clusters, next step is to find the duplicate pages corresponding to these queries. For this purpose, at the back end, *Duplicate Page Extractor* module is made to retrieve a set of search results for each of these queries independently with the help of query processor of search engine and find duplicate page information from this set by applying the proposed ***D****uplicate* ***D****ata* ***D****etection (D3)* algorithm.

$$SS(D_i, D_j) = \frac{\sum_{k=1}^{n} NKW_{k,ti} \times NKW_{k,tj}}{length(D_i) \times length(D_j)} \qquad (4)$$

where t_i and t_j represent the sets containing tokens/ keywords of documents D_i and D_j respectively, n is the number of common tokens in t_i and t_j, $NKW_{k,ti}$ represent the number of occurrences of kth token in D_i.



Algorithm: Favored_Query_Finder( )

```
I/P: A cluster Ci of Queries.
O/P: True or False.

//Start of Algorithm
 1. Queries that are exactly same club them.
 2. Make a set of IP addresses corresponding to the queries.
 3. For (each query $q \in$ Cluster)
       Calculate the Weight of query as:
          $Wt = \frac{\text{No. of IP addresses which fired the query}}{\text{Total no. of IP addresses in that cluster}}$
       If ($Wt$ >= threshold value) then
            Return True;   //Query is considered as favored query
       else
            Return False;   //query is considered as disfavored.
```

Figure 5. Algorithm for Favored Query Finder

The numerator gives the summation of products of term frequencies of common tokens/keywords; e.g. if a common token (say *apple*) appears 3 and 9 times in D_i and D_j respectivey, then one summation term in the numerator would be 3×9.

Length(D_i) represents the length of the document D_i, which can be calculated by (5).

$$length(D_i) = \sqrt{\sum_{k=1}^{N} NKW_{k,ti}^{2}} \qquad (5)$$

where N represent the total number of tokens in D_i, The length is calculated by summation of squares of term frequencies of tokens/keywords in the document. The values for Similarity Score (SS) will come out in the range of [0, 1].

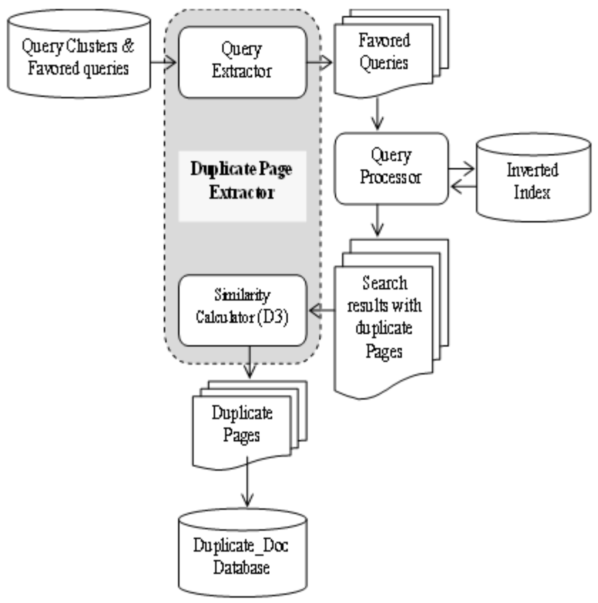


Fig. 6. Working of Duplicate Page Extractor

The *D3* algorithm is described in Fig. 7. It may be noted that if the value of *SS* is above a predefined threshold value (*C*), then documents are considered as near duplicates. The *Duplicate Page Extractor* stores the returned duplicate page information into the *Duplicate_Doc* database along with the ClusterId to which the related query belongs and the PageRank [5] values of pages.

An example *Duplicate_Doc* repository is shown in Fig. 8. Each document *Di* is stored with the associated ClusterId and a list S_i of identical or near identical documents. The list of duplicate pages S_i corresponding to document *Di* can be accessed via the *next* field.

B. *Duplicate Eliminator*

This is a very important module of the framework as it performs online optimization of the search results. It consults the *Duplicate_Doc* repository maintained by the *Duplicate Page Extractor* at the user search time to perform its intended function. The algorithm for this module has been shown in Fig. 9.

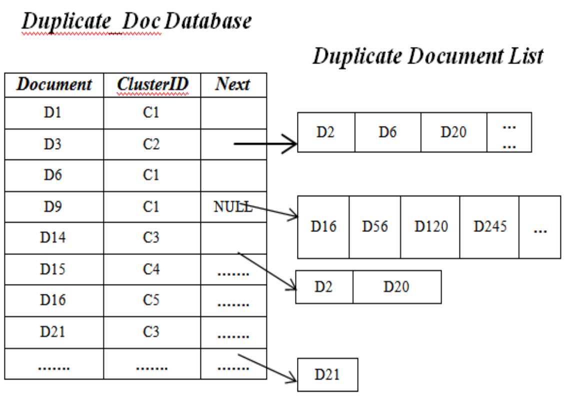
When a user submits a query on the interface of search engine, query keywords are passed to the *Duplicate Eliminator* as well as are given to the Query Processor used to retrieve the matched result pages from the inverted index of the search engine. The eliminator matches query keywords with *KW* set of each query cluster stored in *Query Cluster repository* and the maximum matched clusterID is returned. Now, instead of accessing the whole Duplicate_Doc repository, only the entries with the matched clusterID are accessed for duplicate removal process.

If duplicate documents happen to appear in the results of the user query, the issue is which documents should be kept in the optimized results and which ones should be discarded. The tie-breaker rule adopted by the proposed method is given below, which utilizes the *PageRank* [16] values of the similar web pages:

```
Algorithm: D3()
I/P: Set D of N pages wrt a favored query, similarity threshold C
O/P: Duplicate Documents stored in sets S1, S2, …S_N
// Start of Algorithm
  1. Perform steps 2 for each $D_i \in D$.
  2. Gather the keywords of document $D_i$ from index.
  3. Assign a set Si corresponding to each retrieved
     document $D_i$ that will be used to store similar
     documents.
  4. Initially set $Si=\{D_i\}$
  5. for (each document $D_i \in D$)
       {
        for (each document $D_j$) //$D_j$ are all other docs except $D_i$
          {
            //Calculate Similarity score between Di, & Dj
            $SS(D_i, D_j) = \sum_{i=1}^{n} \frac{KW_{i,t1} * KW_{i,t2}}{\text{length of doc } D_i * \text{length of doc } D_j}$

            If ( SS($D_i$, $D_j$)>= C) then
                $Si = Si \cup D_j$;
          }
       }
  6.  Return S1, S2, …S_N
```

Figure 7. The D3( ) Algorithm



Document	ClusterID	Next
D1	C1	
D3	C2	
D6	C1	
D9	C1	NULL
D14	C3	
D15	C4	…….
D16	C5	…….
D21	C3	…….
…….	…….	…….

Figure 8. Description of Duplicate_Doc Repository

The page with maximum PageRank score will be retained in the final optimized results, while discarding (m-1) duplicates'.

It may be that, when a user submits such query, whose keywords do not match with any cluster present in *Query clusters & favored queries* database, then only the matched results from the search engine index are returned to the interface, while at the same time, the unmatched user query is checked by the Cluster Generator for possible incremental clustering. This new query can either be merged in an existing cluster or assigned a new one.

The Duplicate_Doc database is maintained only for favored queries and their respective clusters as there are maximum chances for the same or similar query to be submitted again by the users. Moreover, considering only favored queries will place less overhead on the normal functioning of a search engine and will also improve the search efficiency in parallel.

IV. Experimental Results

The results of experiments done for duplicate data detection and elimination are presented in this section. The queries under consideration were related to *news* and *media*. Corresponding to each user query, top 100-200 result pages were considered for the experimentation and their associated keywords were stored in MS-Access.

```
Algorithm: Duplicate_Eliminator()
 I/P: A user query q, Duplicate_Doc database, N retrieved
 result pages, Query Cluster & favored queries database.
 O/P: Optimized Result pages (<=N)
 // Start of Algorithm
  1.  Tokenize the query q and place in set KWq;
  2.  Search for the Clusters in query_cluster database,
      which contain the maximum of KWq;
  3.  If (matched cluster found) then GoTo step 4, else
      GoTo step 9.
  4.  Search for the entries Si in Duplicate_Doc repository
      corresponding to the clusterIds found in Step 2;
  5.  Compare the Search Results with the representative
      page of each Si ;
  6.  If (match succeeds with any Si ) then
        (a). Search for the pages ∈ Si in the search results,
        let it be set Si' ;
        (b). From the representative page of Si & pages ∈
        Si', Keep the page with highest PageRank in the
        search results;
        (c). Filter out the pages with low PageRank from
        the search results.
  7.  If any Si still remains to be examined then GoTo
      Step 5, otherwise GoTo Step 8;
  8.  Return optimized result pages.
  9.  Return N result pages without eliminating duplicates
```

Figure 9. The Duplicate Eliminator Algorithm

Near duplicates have been detected efficiently by the proposed *D3()* algorithm. The algorithm was able to find all pairs of web pages, which were duplicated to each other with respect to the predefined threshold values in the range (0.75-0.85).

One example from the Google search results is shown, which indicates the presence of duplicates or near duplicate web documents in the results. For the experimental purposes, the query *q* fired is given below.

q: CBI inquiry sought in Nirupama Pathak Murder Case

Corresponding to this query, the following three URLs dictated the highest similarity to each other, when threshold value of 0.80 was taken:

URL1:*http://timesofindia.indiatimes.com/india/Ambiguities-in-Nirupama-autopsy-report-AIIMS- expert/articleshow/5910173.cms*

URL2:*http://kraran.com/nirupama-pathak-autopsy-report-has-ambiguities/*

URL3:*http://beta.thehindu.com/news/national/article425770.ece*

Similarity Score between Doc1, Doc2 and Doc3, corresponding to URL1, 2 and 3 respectively, using similarity measure (3) is shown in Table I.

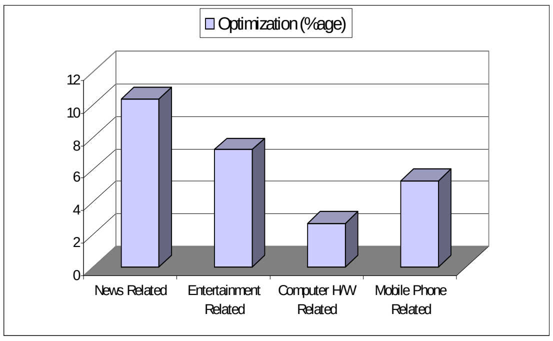


Figure 10. The Optimization of Results with Duplicate Elimination

The above results indicate that URL1 and URL3 are matched to a large extent as compared to URL1 and URL2 or URL2 and URL3. By considering the above said lower threshold measure; it is concluded that the three are near duplicates to each other. Only one URL is retained in the results while discarding the other two according to their PageRanks or Click_Counts (if no PageRank exists).

From the experiments, the resultant performance of search engines in terms of optimization of search results has been shown in Fig. 10. The average values of percentage optimization (reduction) with respect to queries related to different topics is shown in the figure. It may be observed that the queries related to news resulted in pages with up to 5 to 10% pages containing duplicates in the results, therefore removal of such duplicates resulted in optimization (e.g. removal of 8 pages out of 100 resulted in 8% optimization).

TABLE I
SIMILARITY SCORE VALUES FOR SAMPLE DOCUMENTS

Documents	Doc1	Doc2	Doc3
Doc1	1	0.824868	0.868965
Doc2	0.824868	1	0.813696
Doc3	0.868965	0.813696	1

V. CONCLUSION

The explosive growth of web has posed challenges in front of search engines to efficiently retrieve relevant and distinct documents in response to user queries. Document retrieval without duplicates has become a major problem nowadays due to the presence of multiple copies of identical and nearly identical content on the web. The proposed framework addresses this problem with the help of query logs maintained by the search engines. Search result optimization on the basis of usage data proves to be novel in the sense that on submitting the future queries, the distinct pages would be presented and search engine would become adaptive to user needs. The main advantage of the approach is the detection of duplicates without affecting the performance of the search engine as it works in an offline mode. As large repositories would be needed to perform offline filtering of search results, therefore, it is performed online to achieve maximum optimization. The search space can be reduced to a large extent by the proposed mechanism. As a future direction, crawlers of the search engines can be made to utilize the information regarding duplicate pages found by the proposed approach, to enhance the crawler efficiency also.

REFERENCES

[1] Bharat, K., Broder, A.Z, 1999: “Mirror, mirror on the Web: A study of host pairs with replicated content”. In Proceedings of the 8th International World Wide Web Conference (WWW), pp: 501-512.

[2] Xiao, C., Wang, W., Lin, X., Xu Yu, J., 2008. “Efficient Similarity Joins for Near Duplicate Detection”, Proceeding of the 17th international conference on World Wide Web, pp: 131-140.

[3] Claudio Carpineto, Stanislaw Osiñski, Giovanni Romano, Dawid Weiss. “A survey of Web clustering engines”, ACM Computing Surveys (CSUR), Volume 41, Issue 3 (July 2009), Article No. 17, ISSN:0360-0300.

[4] Neelam Duhan, A. K. Sharma, Komal Kumar Bhatia, 2009. “Page Ranking Algorithms: A Survey”, Proceedings of IEEE International Advance Computing Conference, (IACC 2009), pp. 2811-2818.

[5] Lawrance page and sergey Brin, 1998. “The Page Rank Citation ranking: Bringing order to the web”. In Proceedings of seventh web conference (WWW 98).

[6] R. Baeza-Yates, 2004. “Query usage mining in search engines”, Web Mining: Applications and Techniques, Anthony Scime, editor. Idea Group.

[7] BarYossef, Z., Keidar, I., Schonfeld, U., 2007. “Do Not Crawl in the DUST: Different URLs with Similar Text”, 16th International world Wide Web conference, Alberta, Canada,DataMiningTrack,8-12May.